METHOD FOR DETECTING PROGRAM PHASES WITH PERIODIC CALL-STACK SAMPLING

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] Not Applicable.

STATEMENT REGARDING FEDERALLY SPONSORED-RESEARCH OR DEVELOPMENT

The research was sponsored by the United States Government, under contract number NBCH020056. The period of performance was from June 17, 2002, through September 16, 2003. The name of the project is: High Productivity Computing System (HPCS).

[0002]

INCORPORATION BY REFERENCE OF MATERIAL SUBMITTED ON A COMPACT DISC

[0003] Not Applicable.

FIELD OF THE INVENTION

[0004] The invention disclosed broadly relates to the field of computer program execution systems, and more particularly relates to the field of programming language implementation.

BACKGROUND OF THE INVENTION

program. A phase begins when the program calls a method or process and completes when the result of the method is obtained.

[0006] System researchers have studied phase behavior in program execution traces for many years. An example is described in "A study of program locality and lifetime functions," by Denning and Kahn, Proceedings of the Fifth Symposium on Operating Systems Principles, 1975.

[0007] Some previous work has suggested methods wherein if a program execution engine could detect phase shifts, it would exploit the information to improve system performance. Relevant related work is discussed in "Design and Implementation of the Opportunistic Garbage Collector," by Wilson and Moher, in Proceedings of the ACM Conference on Object-Oriented Programming Languages, Systems and Applications, 1989, which describes a system that triggers garbage collection based on a heuristic process that correlates program phases with stack height.

[0008] Several works describe methods whereby an online adaptive optimization system could reset profit data based on automatic phase shift detection. See "Online Feedback-Directed Optimization of Java," Arnold, Hind and Ryder, in Proceedings of the ACM Conference on Object-Oriented Programming Languages, Systems, and Applications, 2002; "Dynamic Hot Data Stream Prefetching for General-Purpose Programs," Chilimbi and Hirzel, in SIGPLAN 2002 Conference on Programming Language Design and Implementation, 2002; "Computing the Similarity of Profiling Data; Heuristics for Guiding Adaptive Compilation," by Kistler and Franz, in Workshop on Profile and Feedback-Directed Compilation, 1998.

[0009] Some of the known art in phase detection techniques for running computer programs looks for patterns in a stream of sequential data, reminiscent to algorithms commonly used in signal processing applications. These techniques require much tuning for sensitivity in edge detection, and rely on numerous heuristic filters to clean up noisy data.

[0010] Instead, it would be highly desirable to provide a method that exploits information that can be gathered by the program execution engine, in order to report phase boundaries definitively, without complications of sensitivity tuning and filter heuristics.

[0011] The previous work, by Wilson and Moher (referenced above), uses heuristics to detect phases of computer programs that provide good opportunities for garbage collection. One method described tags user-interaction routines, and tries to schedule garbage collection during pauses between such routines. The Wilson, et al. paper also suggests taking the height of activation stacks into account. A similar idea, for scheduling garbage collection (GC) based on stack pops, was suggested in "A Real-Time Garbage Collector Based on the Lifetimes of Objects," by Lieberman and Hewitt, Communications of the ACM, 26(6), June 1983.

[0012] These references suggest using stack activation height as an indication of phase behavior. However, a modified program execution engine can further provide information regarding the lifetime of individual activations. There is thus a need for a method to exploit additional information for accurate and simple online phase detection. Further, there is a need for a method that can be implemented with low runtime overhead and can detect nested phases.

SUMMARY OF THE INVENTION

[0013] Briefly according to the invention, a method and system detects phases in running computer programs. The program's runtime representation in memory may include a plurality of stacks wherein each stack comprises a plurality of frames. The method begins by allocating memory space, somewhere in system memory, for an activation counter corresponding to each frame in each stack. Next, the method zeroes the activation counter for each new stack frame wherever the program creates a new stack frame. The method then suspends running of the program at a designated time intervals, and incrementing the activation counter for each frame in each stack. Finally, a phase is associated with an activation whose activation count is non-zero. The system can be a programmable information processing system that comprises program instructions for performing the above method. Alternatively, a system using the invention may comprise one or more application-specific integrated circuits for performing the method.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] FIG. 1 is a flow chart illustrating a method according to the invention.

[0015] FIG. 2, is a flow chart illustrating a garbage collection method according to the invention.

[0016] FIG. 3 is a simplified block diagram of an information processing system according to an embodiment of the invention.

DETAILED DESCRIPTION

[0017] Referring to FIG. 1, there is shown a flow chart illustrating a method 100 for detecting phases in a computer program running at least one thread according to an embodiment of the invention. The method is performed with a system that comprises a plurality of stacks each comprising at least one stack frame. Each stack frame comprises an activation counter. In step 102 an activation count is associated with each frame. Then in step 104 the activation count is zeroed whenever the system creates a new stack frame. Next in decision 106 the system determines whether an interval has transpired during program execution. This can be done using a system clock. If the interval has not transpired the process continues until the interval transpires. Once the interval has transpired, in step 108 the program walks (i.e. examines the content of) each thread's stack and increments the activation count for each frame. At any given time, in step 110 the system associates a phase with an activation whose activation count is zero. The activation count is implemented by reserving storage in each stack frame. The method 100 can further comprise the act of logging activation counts during each interval.

[0018] Once a phase in a program has been detected it is advantageous to schedule performance of certain functions at the end of a phase. Among the functions that can be performed are: (1) comprising scheduling garbage collection after each associated phase; (2) scheduling thread switches at phase boundaries; (3) scheduling checkpoint operations after each associated phase; and (4) presenting a visualization of program phase behavior; (5) resetting profile data at program phase transitions. The

method 100 can also comprise changing the return address to force the program to call a designated procedure when the frame returns.

[0019] There are various ways to implement the activation count. One possibility is to reserve a space for the activation count in the frame data structure. However, it is also possible to store the activation count in a side data structure, different from the frame structure or as an array paralleling the stack.

[0020] We now discuss an embodiment that uses the invention to implement an opportunistic garbage collection algorithm. We assume use of a standard *stop-the-world garbage collector*. The embodiment comprises an implementation that triggers a garbage collection when the following two conditions are met:

a. The heap exceeds a specified occupancy level, and
b. The program exits a relatively long-running phase.

[0021] This embodiment should result in increased garbage collection efficiency, because when a phase ends, the heap should have a favorable ratio of live-to-dead objects. We describe the embodiment in terms of a program with a single running thread. However, extensions for multiple threads are straightforward, and will be apparent to those skilled in the art.

[0022] Referring to FIG. 2, in an embodiment a garbage collection method 200 comprises the following steps. In step 202 the system ensures that each stack frame contains a reserved word, called the *activation count*. In step 204, the system zeroes the activation count whenever it creates a new stack frame. The system keeps a counter, called the *GC counter*, which it initializes to be zero at program start-up. It

resets the *GC counter* to zero after each garbage collection. In step 206, the system chooses a fraction between 0 and 1, called the *long lifetime threshold* and designates this fraction as *L*. After a designated interval 208 (e.g. every 10ms), the system suspends the running thread and performs the following steps: In step 210, it increments the *GC counter*. In step 212, it walks the thread's stack and increments the *activation count* for each frame. In decision 214, the system determines whether the heap occupancy exceeds a designated threshold (e.g. 80%), then in step 216, the system chooses the stack frame *F* with the smallest *activation count*, such that the *activation count* exceeds the *L*GC counter*. Finally, in step 218, if *F* exists, then set state such that when the activation *F* returns, the system invokes garbage collection.

[0023] Referring to FIG. 3 there is shown a simplified block diagram of an information processing system 300 according to an embodiment of the invention. The system 300 comprises a processor (central processing unit) 302, a memory 304, and an Input/output subsystem 306. As shown, the I/O subsystem comprises a compact disk (CD) drive 308 for receiving a CD 310 comprising a computer program product comprising instructions for performing methods according to the invention. However, the I/O subsystem 306 can also be a connection to a network such as a local area network (LAN) or wide area network (WAN), and program instructions can be downloaded from another node (a server) in the network.

[0024] The memory 304 represents both volatile and non-volatile storage. It should be understood that the system 300 may cache certain items of data and instructions in various different parts of memory for performance purposes. The memory comprises a copy of an operating system 312 such as IBM's AIX ™ operating system. A plurality of applications 314 is also stored in memory. The

memory further comprises a stack 316 comprising a plurality of frames (1 through n) and a corresponding plurality of activation counters (AC_1- CA_n).

[0025] The invention can be implemented in various embodiments. The system 300 can be a programmable information processing system, as discussed above, that comprises program instructions for performing the above-described method. Alternatively, a system using the invention may comprise one or more application-specific integrated circuits for performing the method. In most cases, the invention will be implemented as a combination of hardware and software components.

[0026] Therefore, while there has been described what is presently considered to be the preferred embodiment, it will be understood by those skilled in the art that other modifications can be made within the spirit of the invention.

What is claimed is: